Функциональное тестирование Web-приложений на основе технологии UniTesK

А.А. Сортов, А.В. Хорошилов

Аннотация. Статья посвящена анализу дополнительных возможностей автоматизации функционального тестирования Web-приложений на основе технологии UniTesK. В ней рассматриваются существующие подходы к автоматизации функционального тестирования Web-приложений, обсуждаются их достоинства и недостатки. Кроме того, анализируются возможные варианты применения технологии UniTesK для тестирования данного класса приложений, и предлагается способ дополнительной инструментальной поддержки процесса разработки функциональных тестов.

1. Введение

В данной статье Web-приложениями мы будем называть любые приложения, предоставляющие Web-интерфейс. В настоящее время такие приложения получают все большее распространение: системы управления предприятиями и драйверы сетевых принтеров, интернет-магазины и коммутаторы связи – это только небольшая часть приложений, обладающих Web-интерфейсом.

В отличие от обычного графического пользовательского интерфейса Web-интерфейс отображается не самим приложением, а стандартизированным посредником – Web-браузером. Web-браузер берет на себя все взаимодействие с пользователем и обращается к Web-приложению только в случае необходимости. Описание пользовательского интерфейса предоставляется браузеру в стандартном представлении, в роли которого обычно выступает HTML[1] [1].

На рис. 1 представлен процесс работы типичного Web-приложения. Пользователь взаимодействует с приложением посредством Web-браузера, который при необходимости обращается с запросом к Web-приложению, чтобы выполнить ту или иную операцию. Результатом такого обращения является полное или частичное обновление интерфейса приложения, отображаемого в браузере.

[1] Помимо HTML может использоваться XHTML, XML и др.

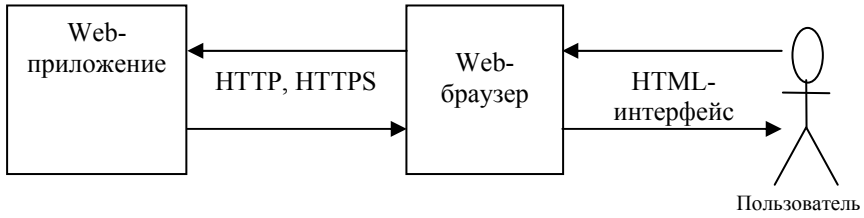

Рис. 1. Процесс работы типичного Web-приложения.

При обращении к Web-приложению браузер посылает запрос по одному из протоколов доступа (HTTP, HTTPS или др.). Web-приложение обрабатывает запрос и возвращает браузеру описание обновленного интерфейса.

Web-приложения в первую очередь характеризуются тем, что их пользовательский интерфейс имеет стандартизированную архитектуру, в которой:

1) для взаимодействия с пользователем используется Web-браузер;
2) взаимодействие с пользователем четко разделяется на этапы, в течение которых браузер работает с одним описанием интерфейса;
3) эти этапы разделяются однозначно выделяемыми обращениями от браузера к приложению;
4) для описания интерфейса применяется стандартное представление (HTML);
5) коммуникации между браузером и приложением осуществляются по стандартному протоколу (HTTP).

Web-приложения можно рассматривать как клиент/серверные приложения, в которых функциональность реализуется как на серверной, так и на клиентской стороне. Функциональность, реализованная на клиентской стороне, как правило, сводится к проверке вводимых данных и реализации дополнительных возможностей интерфейса, что реализуется путем использования скриптовых возможностей, встроенных в HTML (использование Java-script, VBScript и т.д.)[2].

В этой статье мы будем рассматривать функциональное тестирование именно серверной части, оставляя рассмотрение функциональности клиентской части в качестве темы будущих исследований. В этом случае основной интерес представляют взаимодействия браузера с сервером. Эти взаимодействия хорошо формализованы, поскольку осуществляются на основе протокола

[2] Кроме скриптовых возможностей существуют другие средства расширения функциональности, такие как технология ActiveX, Macromedia Flash и др., которые взаимодействуют с Web-приложением, используя свои собственные механизмы. В рамках данной статьи тестирование таких расширений не рассматривается.

HTTP. Четкая формализация взаимодействий может служить основой для автоматизации функционального тестирования.

С другой стороны, представление интерфейса в виде HTML также четко формализовано. Кроме того, в этом описании интерфейса можно выделить действия, приводящие к взаимодействию с сервером. Эти действия связаны с воздействиями на кнопки, активизацией гиперссылок и реакциями на различные события, закодированные в скриптовой части интерфейса. Таким образом, формальное описание интерфейса Web-приложений также предоставляет широкие возможности для автоматизации функционального тестирования.

Но как наилучшим образом использовать потенциал, предоставляемый стандартизированной архитектурой интерфейса Web-приложения? Исследованию этого вопроса и посвящена данная работа.

Статья построена следующим образом. В разделе 2 мы рассмотрим существующие подходы к автоматизации функционального тестирования Web-приложений. В разделе 3 будут представлены основные сведения о технологии автоматизации тестирования на основе моделей UniTesK. Затем мы проанализируем различные варианты моделирования Web-приложений в рамках технологии UniTesK и по результатам этого анализа представим расширение базовой технологии UniTesK, специально адаптированное для функционального тестирования Web-приложений. В заключение, мы рассмотрим пути дальнейшего развития предложенного подхода.

2. Существующие подходы к функциональному тестированию Web-приложений

Самым распространенным является подход, называемый Capture & Playback (другие названия – Record & Playback, Capture & Replay). Суть этого подхода заключается в том, что сценарии тестирования создаются на основе работы пользователя с тестируемым приложением. Инструмент перехватывает и записывает действия пользователя, результат каждого действия также запоминается и служит эталоном для последующих проверок. При этом в большинстве инструментов, реализующих этот подход, воздействия (например, нажатие кнопки мыши) связываются не с координатами текущего положения мыши, а с объектами HTML-интерфейса (кнопки, поля ввода и т.д.), на которые происходит воздействие, и их атрибутами. При тестировании инструмент автоматически воспроизводит ранее записанные действия и сравнивает их результаты с эталонными, точность сравнения может настраиваться. Можно также добавлять дополнительные проверки – задавать условия на свойства объектов (цвет, расположение, размер и т.д.) или на функциональность приложения (содержимое сообщения и т.д.). Все коммерческие инструменты тестирования, основанные на этом подходе, хранят записанные действия и ожидаемый результат в некотором внутреннем представлении, доступ к которому можно получить, используя или распространенный язык программирования (Java в Solex [2]), или собственный язык инструмента (4Test в SilkTest [3] от Segue, SQABasic в Rational Robot [4] от IBM, TSL в WinRunner [5] от Mercury). Кроме элементов интерфейса, инструменты могут оперировать HTTP-запросами (например, Solex [2]), последовательность которых также может записываться при работе пользователя, а затем модифицироваться и воспроизводиться.

Основное достоинство этого подхода – простота освоения. Создавать тесты с помощью инструментов, реализующих данный подход, могут даже пользователи, не имеющие навыков программирования. Вместе с тем, у подхода имеется ряд существенных недостатков. Для разработки тестов не предоставляется никакой автоматизации; фактически, инструмент записывает процесс ручного тестирования. Если в процессе записи теста обнаружена ошибка, то в большинстве случаев создать тест для последующего использования невозможно, пока ошибка не будет исправлена (инструмент должен запомнить правильный результат для проверки). При изменении тестируемого приложения набор тестов трудно поддерживать в актуальном состоянии, так как тесты для изменившихся частей приложения приходится записывать заново.

Этот подход лучше всего использовать для создания прототипа теста, который впоследствии может служить основой для ручной доработки. Одна из возможных доработок – параметризация теста для проверки тестируемого приложения на различных данных. Этот подход называется тестированием, управляемым данными (Data Driven [6, 7]). Основное ограничение – перебираемые данные не должны изменять поведение тестируемого приложения, поскольку проверки, записанные в тестовом сценарии, не подразумевают какой-либо анализ входных данных, т.е. для каждого варианта поведения нужно создавать свой сценарий тестирования со своим набором данных. Некоторые инструменты, реализующие Capture & Playback, предоставляют возможность по перебору данных (например, e-Tester [8] от Empirix); кроме того, над большинством распространенных инструментов существуют надстройки (Convergys Auto Tester [6] - надстройка над WinRunner).

Описанные подходы основываются на построении тестов с использованием тестируемого приложения. В подходе Keyword Driven. [7, 9] предпринимается попытка сделать процесс создания тестов независимым от реализации. Суть подхода заключается в том, что действия, выполняемые в ходе тестирования, описываются в виде последовательности ключевых слов из специального словаря («нажать», «ввести», «проверить» и т.д.). Специальный компонент тестовой системы переводит эти слова в воздействия на элементы интерфейса тестируемого приложения. Таким образом, никакого программирования для создания тестов не нужно. Единственное, что нужно менять при изменении интерфейса, – это компонент, который отвечает за перевод слов из «словаря» в последовательность воздействий на приложение. Комплект тестов может

разрабатываться пользователями, не владеющими навыками программирования, однако для поддержания комплекта в рабочем состоянии программирование все-таки необходимо. В качестве примера инструмента, поддерживающего такой подход к разработке тестов, можно привести Certify [10] от WorkSoft, в котором поддерживается библиотека функций для работы с каждым компонентом интерфейса (окна, гиперссылки, поля ввода и т.д.) и предоставляется язык воздействий на эти элементы (InputText, VerifyValue, VerifyProperty и т.д.).

Основные преимущества этого подхода заключаются в том, что он позволяет создавать тесты, не дожидаясь окончания разработки приложения, руководствуясь требованиями и дизайном интерфейса. Созданные тесты можно использовать как для автоматического выполнения, так и для ручного тестирования.

Основной недостаток этого подхода – отсутствие автоматизации процесса разработки тестов. В частности, все тестовые последовательности разрабатываются вручную, что приводит к проблемам, как на стадии разработки, так и на стадии сопровождения тестового набора. Эти проблемы особенно остро проявляются при тестировании Web-приложений со сложным интерфейсом.

3. Технология UniTesK

Большинство проблем, присущих рассмотренным подходам разработки тестов, решены в технологии UniTesK, разработанной в Институте системного программирования РАН. Технология хорошо себя зарекомендовала при функциональном тестировании разнообразных систем (ядро операционной системы, стеки протоколов, компиляторы). Опыт применения технологии для тестирования Web-приложений показал, что UniTesK может служить хорошей базой для тестирования такого класса приложений. В этом разделе мы остановимся на основных моментах технологии UniTesK (более детальную информацию о ней можно найти в [11, 12, 13, 14]); в последующих разделах рассмотрим особенности применения технологии для тестирования Web-приложений.

Технология UniTesK – это технология разработки функциональных тестов на основе моделей, которые используются для оценки корректности поведения целевой системы[3] и автоматической генерации последовательностей воздействий, далее называемых тестовыми последовательностями.

Оценка корректности поведения целевой системы осуществляется в рамках следующего представления об устройстве интерфейса целевой системы. Предполагается, что целевая система предоставляет набор интерфейсных функций, и все воздействия на нее осуществляются только через вызовы этих

[3] *Целевой системой* мы будем называть программную систему, которая выступает в качестве объекта тестирования. Далее в качестве синонима для термина *целевая система* мы будем также использовать словосочетание *тестируемая система*.

интерфейсных функций. Параметры воздействий, передаваемые целевой системе, описываются входными параметрами интерфейсной функции. Результат воздействия (реакция системы) представляется выходными параметрами, значения которых могут зависеть от истории взаимодействий целевой системы с окружением. Информация об истории моделируется внутренним состоянием целевой системы. Внутреннее состояние влияет на выходные параметры интерфейсных функций и может изменяться в результате их работы.

Следует заметить, что в рамках данной статьи для тестирования Web-приложений рассматривается представление, в котором воздействия на целевую систему и получение ее реакции на это воздействие (выходные параметры интерфейсной функции) рассматриваются как *атомарное* действие. Под атомарностью действия понимается, что следующее воздействие можно произвести только после получения реакции на предыдущее. Технология UniTesK также позволяет представлять целевую систему и как систему с отложенными реакциями, т.е. как систему, разрешающую воздействие до получения всех реакций на предыдущее.

Корректность поведения целевой системы оценивается с точки зрения его соответствия поведению некоторой «эталонной» модели, называемой спецификацией. В технологии UniTesK эталонная модель описывается неявно в виде требований к поведению каждой интерфейсной функции. При задании эталонной модели можно описывать функции и их параметры в достаточно обобщенном виде, отвлекаясь от несущественных подробностей.

Основными компонентами тестовой системы являются итератор тестовых воздействий, оракул и медиатор. Задачей итератора тестовых воздействий, работающего под управлением обходчика, является построение тестовой последовательности, обеспечивающей заранее определенный критерий тестового покрытия. Задачей оракула является оценка корректности поведения целевой системы. Задача медиатора – преобразовывать тестовое воздействие в последовательность реальных воздействий на целевую систему и на основании доступной информации построить новое модельное состояние целевой системы после вызова.

В качестве языка описания компонентов тестовой системы используются спецификационные расширения обычных языков программирования, таких как C# и Java. В этих расширениях реализованы три вида специальных классов, предназначенных для описания компонентов тестовой системы. Из спецификационных классов генерируются оракулы, из медиаторных – медиаторы, а из сценарных – итераторы тестовых воздействий.

В спецификационных классах описываются спецификационные методы, каждый из которых соответствует некоторой интерфейсной функции и содержит формальное описание требований к поведению целевой системы при взаимодействии с ней через данную интерфейсную функцию. Сценарные классы предназначены для описания тестовых сценариев, содержащих

описание единичных воздействий и правил итерации их параметров. Медиаторы генерируются на основе медиаторных классов, которые связывают интерфейсные функции с воздействиями на целевую систему.

Схема работы тестовой системы, разработанной по технологии UniTesK, представлена на рис. 2.

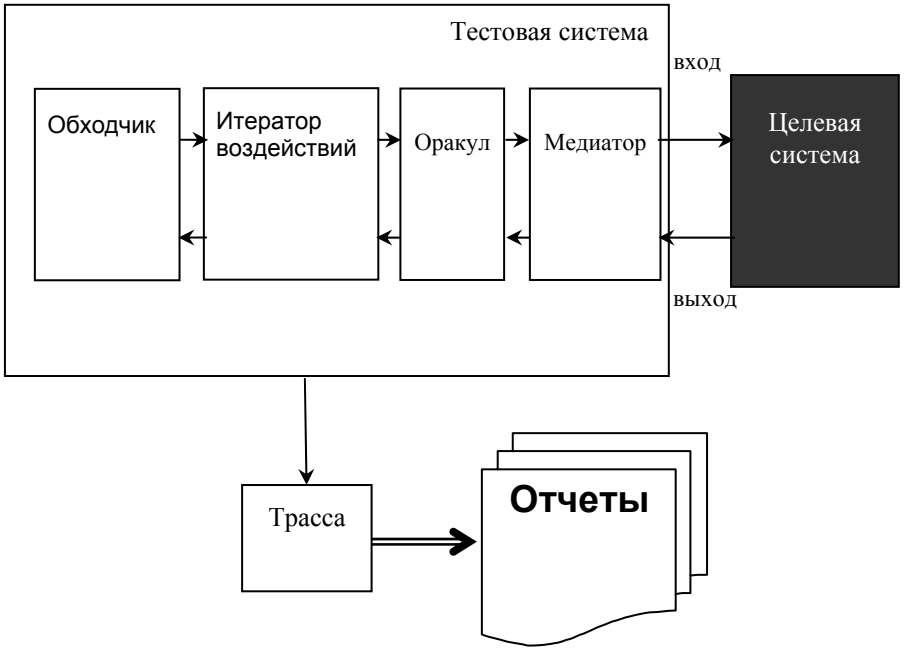

Рис. 2. Схема работы тестовой системы, разработанной по технологии UniTesK.

Основной шаг работы тестовой системы устроен следующим образом. Обходчик выбирает очередное сценарное воздействие. Сценарное воздействие содержит несколько обращений к целевой системе, представляющих собой вызов интерфейсной функции с определенным набором значений входных параметров. Вызов интерфейсной функции передается оракулу, который, в свою очередь, передает его медиатору. Медиатор преобразует вызов интерфейсной функции в последовательность действий над тестируемой системой, получает результат этих действий от тестируемой системы и преобразует его в значения выходных параметров интерфейсной функции. Медиатор также синхронизирует модель состояния тестируемой системы, используемую оракулом для оценки корректности поведения, с ее реальным состоянием.

Оракул, зная значения входных и выходных параметров интерфейсной функции, а также состояние целевой системы, оценивает корректность ее

поведения в рамках данного взаимодействия. Если вердикт оракула является отрицательным, то это значит, что тестовая система нашла несоответствие между поведением целевой системы и требованиями к ней. По завершении выполнения одного шага управление возвращается обходчику, который выбирает следующее сценарное воздействие или принимает решение о прекращении тестирования.

Все события, происходящие в процессе тестирования, находят свое отражение в трассе теста. На основе трассы генерируются различные отчеты, помогающие в анализе результатов тестирования.

Процесс разработки тестов с помощью технологии UniTesK можно представить в виде следующей последовательности шагов[4]:

(1) анализ функциональности тестируемой системы;
(2) формализация требований к функциональности;
(3) связывание формализованных требований с реализацией;
(4) разработка сценариев тестирования;
(5) исполнение тестов и анализ результатов.

Рассмотрим каждый из этих шагов более подробно.

В результате **анализа функциональности** необходимо определить интерфейс тестируемой системы. Для этого требуется выделить функции, предоставляемые системой, и для каждой такой функции определить, что выступает в качестве ее входных и выходных параметров.

На этапе **формализации требований** для каждой интерфейсной функции, выявленной на предыдущем шаге, необходимо описать ограничения на значения выходных параметров в зависимости от значений входных параметров и истории предыдущих взаимодействий с тестируемой системой. Для этого в технологии UniTesK используется широко известный подход программных контрактов [15]. В основе этого подхода лежат инварианты данных, а также предусловия и постусловия интерфейсных операций.

При **связывании требований с реализацией** необходимо описать, как каждая интерфейсная функция отображается на реализацию тестируемой системы. В рамках этого отображения требуется установить правила преобразования вызовов интерфейсных функций в последовательность действий над тестируемой системой, а также правила построения модели состояния тестируемой системы. Для систем с прикладным программным интерфейсом, когда взаимодействие через интерфейсную функцию соответствует вызову функции тестируемой системы, установление такого отображения может быть автоматизировано при помощи интерактивных шаблонов, предоставляемых инструментами семейства UniTesK.

[4] Конечно технология UniTesK не требует строго последовательного выполнения этих шагов. Они описывают только основное направление развития разработки теста. Ничто не мешает возвращаться к предыдущим шагам для уточнения и доработки определенных элементов.

Тестовые сценарии строятся на основе конечно-автоматной модели целевой системы, которая используется для динамической генерации последовательностей тестовых воздействий. Сценарий определяет, что именно рассматривается как состояние автомата, и какие интерфейсные функции с какими наборами аргументов могут быть вызваны в каждом состоянии. Алгоритмы UniTesK обеспечивают вызов каждой интерфейсной функции с каждым набором ее параметров в каждом достижимом состоянии. Для описания сценария необходимо задать способ идентификации состояний и способ итерации входных воздействий в зависимости от текущего состояния. Инструменты семейства UniTesK предоставляют интерактивные шаблоны, которые позволяют упростить разработку тестовых сценариев.

На заключительном этапе технологического процесса происходит выполнение созданных тестов, автоматическая генерация отчетов о результатах тестирования и анализ этих результатов. На основе анализа принимаются решения о создании запросов на исправление дефектов, обнаруженных в тестируемой системе, или о доработке самих тестов с целью повышения уровня покрытия.

4. Применение UniTesK для тестирования Web-приложений

Технология UniTesK применялась для тестирования Web-приложений в нескольких проектах. В ходе разработки тестов выяснилось, что большая часть усилий тратится на создание медиаторов, которые переводят вызов интерфейсных функций в последовательность воздействий на Web-приложение. Анализ опыта показал, что большая часть этой работы может быть автоматизирована, опираясь на стандартизированную архитектуру пользовательского интерфейса Web-приложений. В принципе, эту особенность Web-приложений можно было бы использовать для автоматизации других шагов технологии UniTesK. В этом разделе будут рассмотрены варианты моделирования поведения Web-приложения в контексте возможной автоматизации шагов технологии UniTesK.

Моделирование определяется способом выделения интерфейсных функций и способом построения модели состояния Web-приложения. Первый вариант основывается на стандартном протоколе HTTP, который служит для взаимодействия между Web-браузером и Web-приложением. Поведение Web-приложения рассматривается на уровне HTTP, и этот уровень считается единственно возможным для обращения к Web-приложению. Во втором варианте за основу берется формальное описание интерфейса в виде HTML, которое используется Web-браузером для организации взаимодействия с пользователем. В этом варианте взаимодействие с Web-приложением происходит только посредством Web-браузера. И, наконец, в третьем варианте поведение Web-приложения моделируется без привязки к конкретному способу обращения, основываясь лишь на тестируемой функциональности.

4.1. Моделирование поведения на уровне HTTP

В первом варианте модель Web-приложения представляется одной интерфейсной функцией, описывающей HTTP-запрос к Web-приложению. Параметры этой функции – это параметры запроса (например, тип запроса (GET или POST), адрес (URL), параметры заголовка и т.д.) и список данных, которые передаются Web-приложению. Выходные параметры функции формируются на основе HTTP-ответа, пришедшего от Web-приложения.

Требования к функциональности формулируются в виде набора ограничений на выходные параметры в зависимости от значений входных параметров и модели состояния Web-приложения. Требования в большинстве случаев сильно различаются в зависимости от URL, поэтому спецификацию интерфейсной функции можно разделить на независимые спецификации нескольких интерфейсных функций, каждая из которых описывает поведение, характерное для конкретного семейства значений параметра, определяющего URL. Однако для больших Web-приложений такое описание требований получается громоздким и плохо структурируемым.

Каждая интерфейсная функция соответствует определенному запросу с некоторыми параметрами; в процессе работы тестовой системы вызов интерфейсной функции преобразуется в посылку соответствующего HTTP-запроса серверу. HTTP-запрос строится на основе формальных правил преобразования параметров, поэтому шаг технологии UniTesK, на котором происходит связывание требований с реализацией, полностью автоматизируется.

При создании тестовых сценариев для этого варианта наибольшую трудность представляет организация перебора параметров выделенных интерфейсных функций. Для каждого конкретного случая можно найти наиболее подходящий способ перебора параметров, однако это требует от тестировщика определенной квалификации и опыта.

Следует отметить, что этот вариант позволяет тестировать Web-приложение на устойчивость к некорректным HTTP-запросам, так как можно имитировать ситуации, которые не должны появиться в процессе нормальной работы с Web-приложением посредством браузера.

4.2. Моделирование на уровне Web-браузера

Во втором варианте в качестве интерфейса системы рассматривается интерфейс, предоставляемый Web-браузером. В этом варианте интерфейсным функциям соответствуют воздействия на активные элементы интерфейса, в результате которых происходит обращение к Web-приложению. Такими элементами можно считать гиперссылки, кнопки форм и элементы интерфейса, для которых определена обработка на клиентской стороне, приводящая к обращению к серверу. Входные параметры этих функций – это данные, которые может вводить пользователь, например, при заполнении полей ввода или выборе значений из выпадающих списков и т.д. Таким образом, если

рассматривать HTML-форму, то нажатию на кнопку, по которой отправляются данные, будет соответствовать интерфейсная функция, параметрами которой являются значения полей этой формы.

Стоит оговориться, что в этом варианте в качестве тестируемой системы рассматривается Web-приложение в целом, включая как серверную, так и клиентскую часть. Однако внимание акцентируется на тестировании серверной части, и не ставится задача покрытия функциональности клиентской части.

В этом варианте считается, что выходные параметры интерфейсных функций отсутствуют, поскольку результат воздействия описывается как изменение состояния Web-приложения.

Состояние Web-приложения в этом варианте разбивается на состояние интерфейса, отображаемого Web-браузером, и состояние сервера. К состоянию интерфейса можно отнести текущую отображаемую страницу и состояние элементов на ней. К состоянию сервера относится, например, состояние базы данных, с которой работает Web-приложение, или данные, описывающие сеанс работы пользователя.

Интерфейсные функции доступны не во всех состояниях, так как не во всех состояниях пользовательского интерфейса присутствуют элементы интерфейса, воздействия на которые соответствуют этим интерфейсным функциям. Например, некоторые элементы интерфейса становятся доступны только после авторизации, HTML-формы с полями и кнопками располагаются не на всех страницах Web-приложения. Условия доступности описываются в предусловии и определяются состоянием Web-приложения. Требования к функциональности описываются в постусловии в виде ограничений на состояние, в которое переходит Web-приложение в результате воздействия, описываемого интерфейсной функцией.

Часто одному URL соответствуют пользовательские интерфейсы, содержащие одни и те же наборы интерфейсных функций. В таких случаях эти наборы удобно объединять в группы функций и специфицировать их как методы одного спецификационного класса. В других случаях одни и те же функции могут присутствовать сразу на нескольких интерфейсах, соответствующих разным URL. В этом случае интерфейсные функции удобно объединять в группы в зависимости от функционального назначения и специфицировать отдельно. Это позволяет получить хорошо структурированные спецификации, в которых дублирование описания функциональности сведено к минимуму.

По сравнению с первым, этот вариант позволяет уделить большее внимание описанию именно функциональности Web-приложения, абстрагируясь от деталей обработки HTTP-запросов и ответов, что существенно упрощает моделирование работы пользовательских интерфейсов, обладающих сложным динамическим поведением.

При тестировании вызов интерфейсной функции преобразуется в воздействия на соответствующие элементы пользовательского интерфейса Web-

приложения. Для этого используются специальные средства (например, API Web-браузера), обеспечивающие доступ к внутреннему состоянию Web-браузера и позволяющие моделировать воздействия на элементы интерфейса. Эти же средства используются для получения информации о результатах воздействия, которые отражаются в состоянии пользовательского интерфейса. Такая информация необходима для проверки корректности поведения тестируемого Web-приложения.

В отличие от первого варианта, для определения связи интерфейсных функций с Web-приложением требуются более сложные преобразования, так как обращение к одной интерфейсной функции может быть преобразовано в несколько последовательных воздействий на элементы пользовательского интерфейса. При ручном описании этих преобразований потребуется больше усилий; кроме того, тестировщик должен обладать знаниями и навыками работы со специальными средствами доступа к внутреннему состоянию Web-браузера. Тем не менее, в большинстве случаев этот процесс можно автоматизировать при помощи дополнительной функциональности инструментов тестирования, реализующих технологию UniTesK.

При создании тестовых сценариев для этого варианта удобно опираться на предлагаемый технологией UniTesK метод построения тестовой последовательности, основанный на обходе графа переходов конечного автомата. Часто в качестве состояния автомата удобно рассматривать страницы с одним и тем же URL, а в качестве переходов между состояниями – вызовы интерфейсных функций. При построении тестового сценария основной задачей пользователя становится описание перебора параметров интерфейсных функций. При этом не нужно описывать все интересные пути обхода страниц Web-приложения; обход достижимых состояний автомата будет автоматически реализован при помощи обходчика. Описание перебора параметров интерфейсных функций (т.е. перебор данных на странице, например, перебор значений полей формы) может быть автоматизировано при помощи некоторого интерактивного режима работы пользователя с инструментом. Этот же интерактивный режим работы, в принципе, может быть использован и для создания сценариев тестирования на основе подхода Capture & Playback, если пользователь все-таки захочет вручную выделить некоторые пути обхода страниц.

Итак, во втором варианте пользователю предлагается более «естественный» язык для создания теста – в терминах интерфейсных элементов Web-приложения и воздействий на них. Описания становятся более понятными, что, как следствие, снижает требования к квалификации разработчиков тестов. Недостатком этого варианта по сравнению с первым можно считать отсутствие возможности протестировать работу Web-приложения на некорректных HTTP-запросах.

4.3. Моделирование без привязки к уровню взаимодействия

Третий вариант заключается в рассмотрении только функциональности Web-приложения и абстрагировании от того, каким образом эта функциональность

предоставляется пользователю. В этом случае интерфейсным функциям соответствуют абстрактные действия, которые выделяются на основе требований к функциональности Web-приложения. Одной интерфейсной функции может соответствовать несколько последовательных обращений к серверу (например, авторизация, переход на нужную страницу, заполнение на ней полей формы и нажатие кнопки). Входные и выходные параметры функций также определяются на основе требований и могут быть не привязаны к интерфейсу Web-приложения. Результатом такого подхода являются качественные спецификации с удобным абстрактным интерфейсом, которые можно применять для тестирования на разных уровнях (на уровне EJB, посредством Web-браузера или HTTP-протокола). При этом вся работа по созданию таких спецификаций должна быть целиком возложена на разработчика тестов, поэтому требования к его квалификации в этом варианте значительно выше, чем в первых двух.

Поскольку интерфейсной функции соответствует некоторое абстрактное действие, которое отвечает некоторому варианту работы с Web-приложением, в этом подходе, как и во втором, возникает проблема автоматизации преобразований вызовов интерфейсных функций в обращения к Web-приложению. Эта проблема, так же как и в предыдущем варианте, может быть решена с использованием интерактивного режима работы. Как уже было замечено, спецификацию интерфейсных воздействий можно использовать для тестирования на разных уровнях, поэтому в результате работы пользователя в интерактивном режиме можно сразу получать преобразования в воздействия разных уровней. Интерфейсное воздействие может быть переведено в серию HTTP-запросов или же в последовательность обращений к элементам интерфейса Web-приложения.

При создании сценариев тестирования необходимо задавать перебор параметров интерфейсных функций и, при необходимости, указывать способ определения состояния для работы обходчика. Эти шаги для рассматриваемого варианта не обладают какой-либо спецификой и мало чем отличаются от шагов технологического процесса UniTesK при его традиционном применении.

Подведем итог. Третий вариант моделирования никак не ограничивает разработчика тестов при описании поведения и формализации требований к тестируемой системе. Получившиеся в результате компоненты тестовой системы можно использовать для тестирования на разных уровнях. На шаги технологии в этом подходе практически не влияет специфика Web-приложений, поэтому трудно предлагать какую-либо автоматизацию, за исключением шага связывания формализованных требований с реализацией, на котором разрабатываются медиаторы.

4.4. Сравнение возможной автоматизации вариантов модели-рования

Сравним рассмотренные варианты. Цель сравнения – выбрать вариант для тестирования Web-приложений, который обеспечивал бы наибольшую

автоматизацию процесса разработки тестов при реализации инструментальной поддержки. Вместе с тем, инструментальная поддержка должна обеспечивать высокое качество функционального тестирования на уровне UniTesK, обладая при этом простотой использования и освоения. Каждый вариант мы будем рассматривать, исходя из следующих критериев. Во-первых, рассмотрим каждый вариант с позиции уровня автоматизации каждого шага технологического процесса UniTesK (последний шаг – выполнение тестов и анализ результатов – не рассматривается, так как инструменты UniTesK уже автоматизируют этот шаг в достаточной степени). Во-вторых, рассмотрим уровень абстракции получаемой в каждом варианте модели требований, поскольку более абстрактные модели облегчают повторное использование, а также обеспечивают простоту разработки и сопровождения получаемых тестов. В-третьих, рассмотрим предъявляемые каждым вариантом требования к квалификации пользователя, что обобщает простоту использования и обучения работе с инструментом. Результаты сравнения представлены в Таблице 1.

Вариант моделирования / Критерий	Уровень HTTP	Уровень Web-браузера	Без привязки к уровню взаимо-действия
Автоматизация анализа функциональности тестируемой системы	высокий уровень	высокий уровень	отсутствует
Автоматизация форма-лизации требований	отсутствует	низкий уровень	отсутствует
Автоматизация связывания требований с реализацией	высокий уровень	высокий уровень	низкий уровень
Автоматизация разработки тестовых сценариев	средний уровень	высокий уровень	средний уровень
Уровень абстракции	низкий	средний	высокий
Требования к квалификации разработчика	высокие	средние	высокие

Таб. 1. Сравнение рассмотренных вариантов возможного моделирования Web-приложений с использованием технологии UniTesK.

Прокомментируем результаты, приведенные в таблице. Результатом анализа функциональности тестируемой системы является список интерфейсных функций с входными и выходными параметрами. При моделировании без привязки к уровню взаимодействия этот выбор осуществляется пользователем пол-ностью вручную, исходя из представлений о функциональности тестируемой системы. В других вариантах моделирования можно предложить формальный способ выделения интерфейсных функций на основе анализа интерфейса Web-приложения (может быть, с использованием интерактивного режима).

Формализация требований трудно поддается автоматизации, как и всякий переход от неформального к формальному, но в рамках моделирования на

уровне Web-браузера можно предложить автоматизацию формализации некоторой части требований с использованием интерактивного режима.

Проблема связывания требований с реализацией решается посредством автоматической генерации медиаторов при моделировании как на уровне Web-браузера, так и на уровне HTTP. Для моделирования без привязки к уровню взаимодействия для создания медиаторов можно предложить воспользоваться интерактивным режимом работы с инструментом, при котором для каждой интерфейсной функции определяется последовательность воздействий на элементы пользовательского интерфейса Web-приложения.

При разработке тестовых сценариев можно пользоваться стандартными средствами инструментов семейства UniTesK, которые позволяют автоматизировать процесс задания основных элементов тестового сценария. Однако для моделирования на уровне Web-браузера можно предложить дополнительные средства для организации перебора параметров и идентификации различных тестовых ситуаций, например, извлекать данные для перебора из интерфейса Web-приложения или выбирать элементы интерфейса, идентифицирующие состояние.

При моделировании на уровне HTTP описание функциональности и тестирование осуществляется в терминах HTTP-запросов, что заставляет пользователя разбираться с большим объемом технической информации, повышая тем самым требования к его квалификации. Моделирование на уровне Web-браузера позволяет описывать функциональность и проводить тестирование в терминах элементов пользовательского интерфейса и воздействий на них, что является естественным языком для разработчика тестов. Описание, выполненное в таких терминах, позволяет отразить требования к функциональности на приемлемом уровне абстракции, не сосредотачиваясь на деталях технологий, лежащих в основе Web-приложения. Третий вариант не ограничивает пользователя в свободе выбора уровня описания функциональности, но вместе с тем не предоставляет дополнительных возможностей по автоматизации, возлагая всю работу на пользователя, что требует от него определенных навыков и опыта.

Описанные варианты моделирования были опробованы в ходе тестирования Web-приложений с использованием технологии UniTesK. Все шаги технологического процесса были реализованы с использованием инструментов UniTesK без дополнительной автоматизации, которая упоминалась в обзоре вариантов моделирования. Анализ процесса разработки показал необходимость и подтвердил выводы о возможности дополнительной автоматизации шагов технологии. Кроме того, опыт передачи тестовых наборов, разработанных с помощью технологии UniTesK, в реальное использование показал, что моделирование без привязки к уровню взаимодействия требует от разработчиков и последующих пользователей хорошего знания технологии UniTesK. В то же время, моделирование на

уровне Web-браузера более естественно воспринималось пользователями, тестирующими Web-приложения и не владеющими технологией UniTesK.

Оценивая рассмотренные варианты с этих позиций, можно сказать, что моделирование на уровне Web-браузера является наиболее подходящим для дальнейшей автоматизации и дополнительной инструментальной поддержки. В следующем разделе мы остановимся более подробно на реализации дополнительной инструментальной поддержки этого подхода.

5. Дополнительная инструментальная поддержка

Основной задачей, возлагаемой на инструментальную поддержку, является упрощение работы пользователя по созданию компонентов тестовой системы. Это достигается за счет дополнительной автоматизации шагов технологического процесса UniTesK с учетом специфики Web-приложений. Первый шаг технологического процесса UniTesK – анализ функциональности тестируемой системы – не предполагает инструментальной поддержки, однако для Web-приложений можно предложить способ выделения интерфейсных функций на основе автоматизированного анализа интерфейса Web-приложения. На шаге формализации требований пользователь может описывать требования в виде условий на различные атрибуты элементов интерфейса; эти условия могут строиться с использованием поддержки инструмента. Информации, собранной при автоматизации первого и второго шагов, оказывается достаточно для автоматического связывания интерфейсных функций с Web-приложением. Для шага разработки тестовых сценариев предлагаются дополнительные возможности по описанию его компонентов в терминах интерфейса Web-приложения. Последний шаг не требует дополнительной автоматизации, так как все инструменты семейства UniTesK уже предоставляют развитые средства выполнения тестов и анализа их результатов.

При использовании дополнительной инструментальной поддержки процесс разработки тестов для функционального тестирования Web-приложений изменяется, и состоит из следующих шагов:

1) создание модели Web-приложения;
2) создание тестового сценария;
3) выполнение тестов и анализ результатов.

Первый шаг – создание модели Web-приложения – включает в себя определение интерфейсных функций, описание требований к ним и их связывание с Web-приложением, т.е. объединяет первые три шага технологии UniTesK. Основная задача этого шага – формализация требований к интерфейсным функциям – в отличие от второго шага технологии UniTesK может быть частично автоматизирована, а выделение интерфейсных функций и их связывание с Web-приложением происходит автоматически. Два последних шага соответствуют двум последним шагам технологии UniTesK и отличаются только уровнем автоматизации.

Рассмотрим более подробно перечисленные выше шаги.

На первом шаге должно быть получено описание модели, состоящее из набора интерфейсных функций и описания требований к ним. Интерфейсная функция соответствует воздействию на интерфейс Web-приложения, в результате которого происходит обращение к серверу. Элементы интерфейса, влияющие на параметры этого обращения, включаются в список параметров интерфейсной функции. Результатом воздействия является обновление интерфейса, которое описывается в требованиях к интерфейсной функции. Например, для HTML-формы регистрации можно описать интерфейсную функцию, соответствующую отправке данных формы на сервер, параметрами которой являются значения полей, входящих в форму. В описание требований включается информация о значениях, для которых успешно выполняется регистрация, и описываются ограничения на состояние обновленного интерфейса.

Разбиение множества всех возможных состояний интерфейса Web-приложений на классы эквивалентности представляется в модели набором страниц. Это разбиение, которое, с одной стороны, используется при описании требований, с другой стороны, является основой для определения состояния в сценарии. По умолчанию разбиение на страницы осуществляется по адресу (URL) HTML-документа, отображаемого в Web-браузере. Однако пользователь может переопределить разбиение произвольным образом. При традиционном способе построения Web-приложения, когда для каждого URL определяется его собственный интерфейс, разбиение по умолчанию соответствует представлению пользователя о тестировании Web-приложения – пройти все страницы и проверить всю возможную функциональность на каждой из них.

Также естественным для пользователя требованием к Web-приложению является требование перехода с одной страницы на другую в результате активизации гиперссылки или нажатия на кнопку HTML-формы. Такие требования легко описываются с помощью понятия страниц. В более сложных случаях, например, для описания требования к результату работы HTML-формы поиска по некоторому критерию, требования формулируются в виде условий на атрибуты элементов интерфейса.

Автоматизация построения модели поддерживается в процессе сеанса работы с тестируемым Web-приложением. Пользователь осуществляет навигацию по страницам приложения, редактируя список интерфейсных функций и их параметров, который автоматически предлагается инструментом, и добавляет описания требований, формулируя их в виде проверки некоторых условий на атрибуты элементов интерфейса. Для формулировки проверок инструмент предоставляет возможность выделения интерфейсных элементов и задания условий на их атрибуты.

В результате этого шага инструмент создает из модели Web-приложения компоненты тестового набора UniTesK, обычно появляющиеся на первых трех шагах технологии – это спецификационные классы, описывающие

интерфейсные функции, и медиаторные классы, реализующие связь между интерфейсными функциями и тестируемой системой.

В спецификационных классах для каждой интерфейсной функции создаются спецификационные методы, в которых описываются требования к поведению функций, сформулированные при работе инструмента. Для этого используются предусловия и постусловия спецификационных методов. В том случае, если средств, предоставляемых инструментом, недостаточно для описания функциональности Web-приложения, полученные спецификационные классы могут быть доработаны вручную.

В медиаторных классах описывается связь между спецификацией и тестируемой системой. Для каждого спецификационного метода задается медиаторный метод, который преобразует вызов этого спецификационного метода в соответствующее воздействие на интерфейс Web-приложения. Это преобразование осуществляется следующим образом. Для каждого параметра спецификационного метода медиатор находит соответствующий ему элемент интерфейса Web-приложения и устанавливает значение его атрибутов в соответствии со значением параметра. Затем медиатор осуществляет воздействие требуемого типа на элемент интерфейса, соответствующий данной интерфейсной функции, и ожидает реакции Web-приложения. Как правило, реакция на воздействие заключается в обращении Web-браузера к Web-серверу и получении от него нового описания интерфейса. Медиатор дожидается завершения обновления состояния интерфейса и синхронизирует состояние модели.

На втором шаге нужно получить описание тестов для Web-приложения. При создании тестов используется подход, предлагаемый технологией UniTesK. Согласно этому подходу тесты описываются в виде тестовых сценариев, в основе которых лежит алгоритм обхода графа переходов конечного автомата. Для каждого тестового сценария нужно выбрать подмножество интерфейсных функций, для тестирования которых предназначен данный сценарий. Для каждой выбранной функции нужно задать правила, по которым будут перебираться ее параметры. Кроме того, нужно задать правила идентификации состояний тестового сценария.

Для автоматизации процесса создания тестового сценария предоставляется возможность определять итерацию для параметров выбранных интерфейсных функций на основе готовых вариантов перебора. Для этого могут использоваться библиотечные итераторы и итераторы, разработанные пользователем. Данные, которые вводились в ходе сеанса работы с инструментом на первом шаге, также могут быть включены в качестве дополнительных значений для заданной итерации. Кроме того, инструмент может предложить перебор параметров, построенный на основе анализа интерфейса Web-приложения. Например, использовать для итерации значения элементов выпадающего списка или же значения, которые берутся из разных интерфейсных элементов, например, расположенных в столбце некоторой таблицы.

Для решения другой задачи, связанной с определением состояния тестового сценария, инструмент предоставляет средства для описания этого состояния в терминах интерфейса Web-приложения. В качестве состояния по умолчанию инструмент предлагает использовать страницу HTML-документа. Для более детального разбиения пользователю предоставляется возможность уточнить описание состояния, выбрав элементы интерфейса и указав условия на их атрибуты или на наличие этих элементов в интерфейсе. Кроме того, для описания состояния можно пользоваться описанием состояния сервера.

Следует заметить, что информации, собранной на шаге построения модели Web-приложения, уже достаточно для создания тестового сценария. При создании тестового сценария по умолчанию используются все выделенные интерфейсные функции, для итерации параметров которых используются итераторы по умолчанию и значения, введенные пользователем в ходе сеанса построения модели, а в качестве состояния сценария используется страница HTML-документа.

В качестве альтернативного способа создания тестов для Web-приложения можно использовать подход Capture & Playback. В процессе работы пользователя с Web-приложением инструмент записывает последовательность воздействий на интерфейс, на основе которой генерирует последовательность вызовов интерфейсных функций, соответствующую записанным воздействиям.

Итак, по сравнению с базовым подходом UniTesK описанный подход обладает следующими преимуществами. Во-первых, уменьшается объем ручного труда за счет автоматизации действий, предписываемых технологией UniTesK. Во-вторых, снижаются требования к квалификации пользователей технологии, так как в этом подходе основным языком взаимодействия с пользователем является не язык программирования (или его расширение), а язык элементов интерфейса и воздействий на них. Следует заметить, что этот подход сохраняет большинство преимуществ технологии UniTesK – гибкую архитектуру тестового набора, обеспечивающую возможность повторного использования компонентов, автоматическую генерацию тестовых последовательностей.

Недостатком данного подхода является отсутствие возможности создания тестов до появления реализации, поскольку подход основан на использовании реализации для дополнительной автоматизации шагов технологии UniTesK. Однако наличия прототипа уже достаточно для начала процесса разработки тестов.

6. Сравнение с другими подходами

В предлагаемый подход, по возможности, были включены достоинства распространенных подходов и инструментов, предназначенных для функционального тестирования Web-приложений. Этот подход, как и другие, позволяет строить тесты, оперируя терминами интерфейса и действий с ним; полученные тесты могут быть расширены с использованием итерации данных. Предоставляются средства для автоматического прогона полученных тестов, анализа их

результатов и генерации отчетов о покрытии и обнаруженных ошибках. Инструмент, реализующий данный подход, может быть использован для создания тестов в стиле Capture & Playback с сохранением всех достоинств этого подхода к тестированию. В реализации описанного подхода также присутствует и основное достоинство подхода Keyword Driven – хорошая архитектура тестового набора, обеспечивающая устойчивость при изменении интерфейса.

В то же время предложенный подход отличается от других рядом преимуществ. В отличие от подхода Capture & Playback он позволяет автоматически генерировать тестовые последовательности, покрывающие различные тестовые ситуации. Таким образом, в этом подходе затраты на создание комплекта тестов того же качества становятся меньше. Другим достоинством данного подхода является возможность создания тестов для не полностью корректной реализации. Если в реализации присутствует ошибка, заключающаяся в том, что при некотором воздействии приложение переходит на некорректную страницу, то инструмент, реализующий Capture & Playback, ошибочно запоминает результат этого перехода как правильный. Добавление этого теста в автоматизированный тестовый набор будет возможно только после исправления ошибки – нужно, чтобы инструмент запомнил корректный результат. В описанном же подходе при описании модели можно явно указать ожидаемую страницу и ее параметры, после чего инструмент будет рассматривать любой другой результат как ошибочный и генерировать соответствующие отчеты об ошибках.

Подход Keyword Driven не предоставляет возможности автоматизации разработки тестов и хорошо автоматизирует только выполнение тестов. Описанный подход автоматизирует как выполнение, так и разработку тестов.

К перечисленным преимуществам можно добавить возможность автоматизированного анализа интерфейса Web-приложения, благодаря чему пользователь может выбирать интерфейсные функции из списка функций, автоматически найденных на странице. Кроме того, в дополнение к традиционным источникам данных для итерации параметров, таких как файлы различных форматов и базы данных, инструмент позволяет извлекать данные из интерфейса Web-приложения, что удобно использовать для организации динамически изменяемого перебора параметров.

7. Направления дальнейшего развития.

В данной работе мы представили расширение технологии UniTesK, которое автоматизирует процесс создания тестов для тестирования функциональности Web-приложений. Тесты строятся в терминах элементов интерфейса и воздействий на них в процессе интерактивной работы пользователя с инструментом, реализующим это расширение. Инструмент взаимодействует с Web-приложением, автоматизируя анализ его интерфейса в предположении, что основная функциональность Web-приложения реализована на стороне сервера.

В заключение рассмотрим возможные направления развития данного подхода. Одним из наиболее важных направлений является автоматизация поддержки тестового набора в актуальном состоянии при изменении интерфейса Web-приложения. Эта проблема присутствует во всех подходах, в которых тесты строятся на основе уже работающей реализации. Единственное относительно успешное решение заключается в локализации компонентов теста, зависящих от деталей реализации интерфейса Web-приложения, позволяющей сократить усилия по приведению тестов в соответствие с изменившимся интерфейсом.

Следующее направление – тестирование функциональности, реализованной в пользовательском интерфейсе. Напомним, что эта функциональность обычно заключается в проверке корректности входных данных и реализации дополнительных возможностей интерфейса. Для описания этой функциональности нужно использовать принципы выделения интерфейсных функций, основанные на более детальном описании взаимодействия пользователя с Web-браузером.

И, наконец, можно выделить направление, связанное с расширением видов тестирования, поддерживаемых инструментом, начиная от генерации некорректных HTTP-запросов для тестирования устойчивости Web-приложения и заканчивая использованием разработанных тестовых наборов для нагрузочного тестирования.

Литература

1. D. Raggett, A. Le Hors, I. Jacobs. HTML 4.0 Specification URL: http://www.w3.org/TR/html40
2. http://solex.sourceforge.net
3. http://www.segue.com
4. http://www-306.ibm.com/software/awdtools/tester/robot
5. http://www.mercury.com/us/products/quality-center/functional-testing/winrunner
6. Shakil Ahmad. Advance Data Driven Techniques
7. Keith Zambelich. Totally Data-Driven Automated Testing URL: http://www.sqa-test.com/w_paper1.html
8. http://www.empirix.com
9. Carl J. Nagle. Test Automation Frameworks. URL: http://safsdev.sourceforge.net/DataDrivenTestAutomationFrameworks.htm
10. http://www.worksoft.com
11. I.B. Bourdonov, A.S. Kossatchev, V.V. Kuliamin, A.K. Petrenko. UniTesK Test Suite Architecture. Proc. of FME 2002. LNCS 2391, pp. 77-88, Springer-Verlag, 2002.
12. В.В. Кулямин, А.К. Петренко, А.С. Косачев, И.Б. Бурдонов. Подход UniTesK к разработке тестов. Программирование, 29(6):25–43, 2003.
13. http://www.unitesk.com
14. А.В. Баранцев, И.Б. Бурдонов, А.В. Демаков, С.В. Зеленов, А.С. Косачев, В.В. Кулямин, В.А. Омельченко, Н.В. Пакулин, А.К. Петренко, А.В. Хорошилов. Подход UniTesK к разработке тестов: достижения и перспективы. Труды Института системного программирования РАН, №5, 2004. http://www.citforum.ru/SE/testing/unitesk
15. Bertrand Meyer. Applying 'Design by Contract'. IEEE Computer, vol. 25, No. 10, October 1992, pp. 40-51.